HESPIDS: A Hierarchical and Extensible System for Process Injection Detection using Sysmon

Rodney Thomas
University of Idaho
rwthomas@uidaho.edu

Stu Steiner
Eastern Washington University
ssteiner@ewu.edu

Daniel Conte de Leon
University of Idaho
dcontedeleon@ieee.org

Abstract

Advanced Persistent Threat (APT) actors are increasingly utilizing Living-off-the-Land (LotL) cyber attack techniques to avoid detection. LotL are techniques that abuse legitimate functionality to perform malicious cyber activities. A common LotL cyber attack technique, that is currently very difficult to detect and prevent, is malicious process injection, MITRE ATT&CK ID: T1055. We report on the initial results for HESPIDS: A Hierarchical and Extensible System for Process Injection Detection using Sysmon. We developed a hierarchical graph-based detection approach for accurate and automated detection for five process injection techniques in Windows. These techniques include four of eleven T1055 sub-techniques: DLL Injection, PE Injection, APC Injection, Process Hollowing, plus API Hooking (T1056.004). Our novel detection approach exhibits, within the limitations of our small testing environment, very high sensitivity and specificity. HESPIDS demonstrates a promising avenue for development of automated detection of advanced cyber threats.

1. Introduction

Advanced Persistent Threat (APT) groups are increasingly utilizing an attack technique known as Living off the Land (LotL). LotL is a type of cyber attack technique in which legitimate system functionality is abused to perform unauthorized or malicious cyber activities. The SolarWinds attack, first discovered in December 2020 [5], used several attack techniques that fall within the LotL category.

LotL techniques allow malicious actors success at evading detection when executing unauthorized activities. LotL cyber attacks are sometimes referred to as zero footprint attacks, since the attack does not require the installation of malware on the compromised system.

1.1. Current Problem

LotL attacks are very difficult to detect and prevent because they abuse functionality that is used in the system as part of its expected operation, which makes separating authorized from unauthorized operations extremely difficult. This difficulty, in detecting LotL attacks, causes delays in discovery and mitigation because security operation center (SOC) threat analysts have to manually investigate these events.

The MITRE ATT&CK framework [22] is a knowledge base of adversary tactics and techniques that provides common cyber attacks nomenclature. MITRE ATT&CK defines Process Injection (ID: T1055) as "*A method of executing arbitrary code in the address space of a separate live process [22].*" T1055 defines 11 sub-techniques. Process Injection is one of the most commonly used LotL cyber attack techniques.

1.2. Contribution

Our objective was to empower threat analysts by investigating automating the detection of LotL Process Injection attacks in Windows. We believe we succeeded through the design of HESPIDS: A Hierarchical and

Table 1. Mitre att&ck: process injection (T1055) sub-techniques plus sub-technique T1056.004

ID	*Sub-technique Name*
T1055.001	**Dynamic-link Library Injection**
T1055.002	**Portable Executable Injection**
T1055.003	Thread Execution Hijacking
T1055.004	**Asynchronous Procedure Call**
T1055.005	Thread Local Storage
T1055.008	Ptrace System Calls (Linux)
T1055.009	Proc Memory (Linux)
T1055.011	Extra Window Memory Injection
T1055.012	**Process Hollowing**
T1055.013	Process Doppelgänging
T1055.014	VDSO Hijacking
T1056.004	**Credential API Hooking**

URI: https://hdl.handle.net/10125/80247
978-0-9981331-5-7

Extensible System for Process Injection Detection using Sysmon. HESPIDS uses a hierarchical graph-based detection approach that exhibits accurate detection for five sub-techniques, shown in bold in Table 1.

1.3. Overview of the Rest of this Article

The rest of this article is organized as follows: Section 2 provides a review of related work. Section 3 details the development and evaluation environment. Section 4 describes our automated detection approach and corresponding experiments and results. Section 5 presents our conclusion and potential future work.

2. Related Works

Published research concerning the detection of process injection attacks in Windows is very limited. Current solutions focus on malware detection approaches not capable of adequately detecting process injection and other LotL cyber attack techniques. In this section we present the most closely related works and a brief comparison with HESPIDS.

2.1. Related Academic Contributions

Mavroeidis and Jøsang: Research in 2018 by Mavroeidis and Jøsang [8] proposed a "*...cyber threat intelligence ontology (CTIO) rich enough for consuming and representing information from several different sources, such as taxonomies, sharing standards, and domain expertise with the purpose of supporting decision making*" [8].

Approach: Mavroeidis and Jøsang [8] used Sysmon Aggregator to aggregate Windows event logs. Then they passed a combined file to a parsing engine which passed the results to a lookup engine; the lookup engine queried a database. The results of the query were passed to a SPARQL engine which produced a Resource Description Framework (RDF) output. This RDF coupled with the threat information sharing engine, produced a threat classification.

Prediction Model: Based on the resulting RDF and the threat information sharing engine, the decision making engine provided a threat classification of High, Medium, Low, or Unknown.

Mikhail et al.: In 2020, Mikhail et al. attempted to answer the question "*Can a framework be developed for non-data scientists to determine whether a given adversary technique is best detected with a heuristic analytic or a machine learning (ML) analytic?*" [11]

Approach: Mikhail et al. consolidated multiple system events into a single process event. The single process event was then used to generate analytics using a tree ensemble machine learning model. Data was gathered by deploying six host sensors in the MITRE production network. These sensors collected data for approximately two weeks.

Prediction Model: Based on the values from the tree ensemble model the highest scoring single path from all the trees was extracted. Mikhail et al. state that the presented approach "*... can detect multiple variations of a single attack technique by capturing and generalizing system behaviors [11].*"

Sun et al.: In 2021 by Sun et al. [23] investigated the DNS activities of malware injected processes versus the DNS activities of Program-DNS processes.

Approach: Sun et al. [23] analyzed two factors. The first factor was `consistency ratio` which measured the similarity of a DNS request by a process to DNS requests for similar processes. The second factor was `domain types` where the types of the domains queried were captured.

Prediction Model: The prediction model consisted of a machine learning algorithm. There were two datasets for training data for the classifier and two additional detection datasets for training data.

Myllyla and Costin: In 2020 by Myllyla and Costin's research focused on the pitfalls concerning failed detection, including attempting to define their root causes [12].

Approach: After careful analysis Myllyla and Costin created detection logic and rules to search log queries from an SIEM.

Prediction Model: Myllyla and Costin's prediction model used custom search queries and Sigma rules to detect previous undetected malicious processes.

HESPIDS: Compared to related articles HESPIDS shares some similarities such as event collection and log aggregation. However HESPIDS is unique in its graph-based prediction model and its implementation using current open source tools.

Log Analysis: HESPIDS automatically aggregates the Sysmon logs by forwarding the logs using Windows Event Forwarding. The log collector then forwards the aggregated logs using Winlogbeat to the Hunting Elasticsearch Logstash Kibana (HELK) stack. The HELK stack uses KSQL to implement a decision graph.

Prediction Model HESPIDS prediction engine uses a dynamic decision graph implemented using KSQL queries. The result of these queries, which are run dynamically, periodically, and automatically classify sets of observed events as potentially malicious. The detection of a malicious process injection attack is then displayed on the SOC analyst's screen.

2.2. Other Related Work

Web searches for malicious process injection detection in Windows result in very few significant works or resources. Two works that standout on this area are: The CyberWarDog blog by Roberto Rodriguez [18] and the article in *Ten Process Injection Techniques* by A. Hosseini [7]. Rodriguez's blog provides several detailed articles on how to detect malicious process injection in Windows using Sysmon. Hosseini's article is very likely the most detailed account and classification of process injection techniques in Windows.

Furthermore, to investigate the state-of-the practice, we installed the latest available version of Security Onion [3]. Security Onion is an open source security operations center (SOC) toolset. It includes full support for Plays (pre-encoded SOC level alert rules) written in the Sigma alert rules language and all 265 Sigma Community rules preinstalled [14]. Within this set of 265 community rules, a search for T1055 returns seven plays. These seven plays are for generating alerts targeted to specific indicators of compromise for specific malware or known exploits, for example, CVE-2019-1378 or Dridex. A search for T1056 returns zero plays. These seven plays are extremely specific which makes them less susceptible to false positives but also susceptible to bypass.

The Sigma rules language specification and associated rule rewriting plugins [14] do support sharing pre-encoded alerts. However, the Sigma language was developed as a way to encode and share Security Information and Event Management (SIEM) alerts. SIEM alerts are focused on string pattern matching rather than the higher level correlations needed for threat hunting.

Moreover, The MITRE ATT&CK page on available mitigation techniques for process injection attacks in Windows (M1040) provides a generic mitigation strategy and references just two articles. The first is an article by Microsoft [10] on how to enable a set of Attack Surface Reduction (ASR) rules in Windows. The second is an article by M. Nelson [13] describing how to mitigate DDE-based attacks and a mitigation for the specific case of Excel being injected through OneNote via DDE.

It's possible commercial SOC toolsets contain other predefined and pre-encoded techniques for automatically and accurately detecting process injection attacks in Windows. However, if these techniques exist, they do not appear to be shared widely with the SOC and threat hunting communities. Common languages for specifying and sharing high-level threat hunting tasks also appear nonexistent.

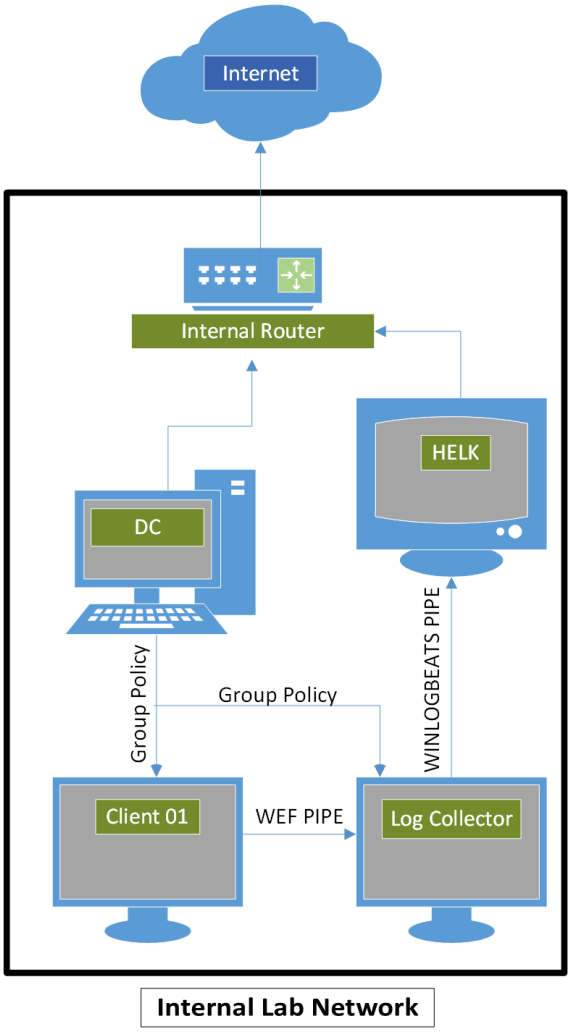

Figure 1. HESPIDS virtual laboratory environment

3. Development/Evaluation Environment

Using a powerful workstation we created a simulated organizational network and mini-SOC using virtual machines (VM) and virtual networks. The workstation hardware was an AMD Ryzen Threadripper 1950X (32 vCores) CPU, 64 GB RAM, and 1 TB NVMe. The workstation software was Windows 10 Edu and VMWare Workstation. The virtual machines in this environment and their network connections are depicted in Figure 1 and described below.

We used this environment to: (1) carry out simulated process injection attacks, using Atomic Red [17] and InjecProc [15]; (2) collect, forward, and store log/event data; (3) search and analyze log/event data; (4) develop and test our detection; and (5) verify and visualize detection results.

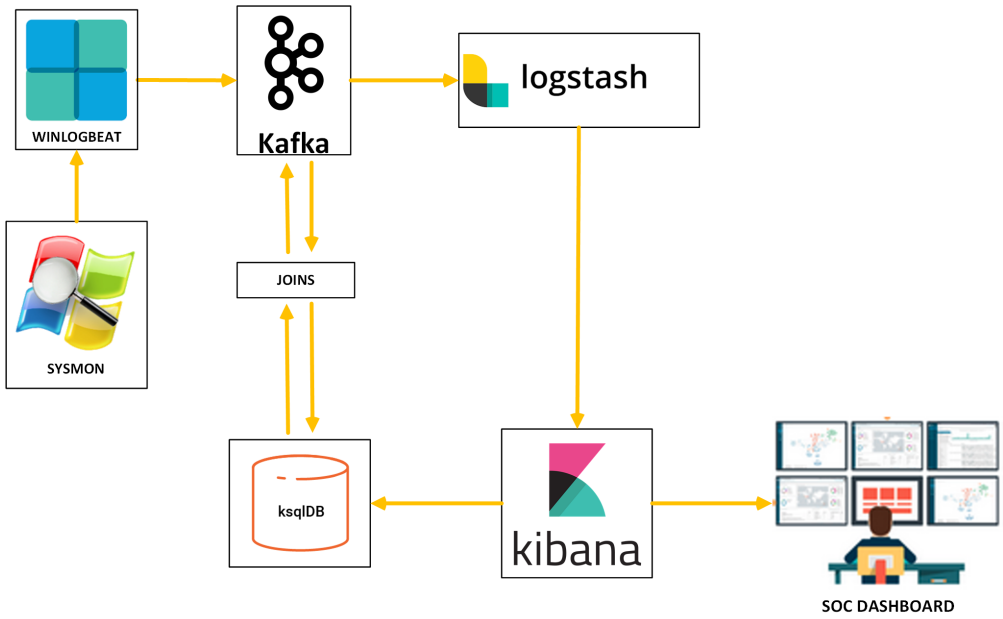

Figure 2. Sensing and detection data flow for our mini-SOC development and evaluation platform

1. `DC`: Domain Controller; Windows Server 2019 VM; vRAM: 8GB, vDisk: 100GB, vCPU: 02. This VM simulates an enterprise Windows domain controller that manages authentication and group policies [9]. Implemented policies require all client PCs (`Client 01`) Windows logs be automatically forwarded to the `Log Collector` every minute (rate is configurable).

2. `Log Collector`: Windows Server 2019 VM; vRAM: 16GB, vDisk: 100GB, vCPU: 08. Its role is collecting Windows logs from `Client 01` and forward to the mini-SOC (`HELK`) using *Winlogbeat* [4].

3. `HELK`: SOC analytics; Ubuntu 20.04 VM; vRAM: 16GB, vDisk: 100GB, vCPU: 08. This VM was built based on a customized version of the Hunting Elasticsearch Logstash Kibana (HELK) toolset [19]. HELK is an open source toolset for security analytics and threat hunting. It uses Apache Kafka which is an event streaming and analytics platform supporting the KSQL stream-oriented query language [1]. This customized environment enabled live data analytics of Sysmon events collected from the Windows client VM (`Client 01`). All internal VM components were run using Docker.

4. `Client 01`: Windows 10 Edu VM; vRAM: 16GB, vDisk: 100GB, vCPU: 08. This client was fully patched and joined to the domain implemented by `DC`. This client PC was the target of the simulated process injection attacks. It had installed Sysmon with a tailored configuration.

5. `Internal Router`: pfSense VM; vRAM: 4GB, vDisk: 40GB, vCPU: 1. This VM simulated an edge router/gateway.

HESPIDS' implementation environment relies on the HELK stack and Kafka, coupled with Windows Sysmon, Windows Event Forwarding (WEF) and Winlogbeats. These applications all work together to collect and aggregate logs and implement the detection graphs. The output of the detection graphs, implemented using KSQL, may be shown in a Kibana dashboard. Then a security analyst can connect to the environment and visualize the results of the automated threat classification, as shown in Figure 3.

3.1. Software Components and Data Flow

Figure 2 illustrates HESPIDS' threat intelligence architecture; flow of data; detection query creation; and visualization and analytics. Once the `Sysmon` data is passed from the `Log Collector` machine to the `HELK` machine, using the `Winlogbeat` stream, the HELK stack controls processing the data.

The Kafka Broker controls the flow of the data. The Kafka broker and the `KSQL` database perform a series of joins on the data to determine if a malicious process injection is occurring. Once the data is returned to the Kafka broker from the `KSQL` database, the broker

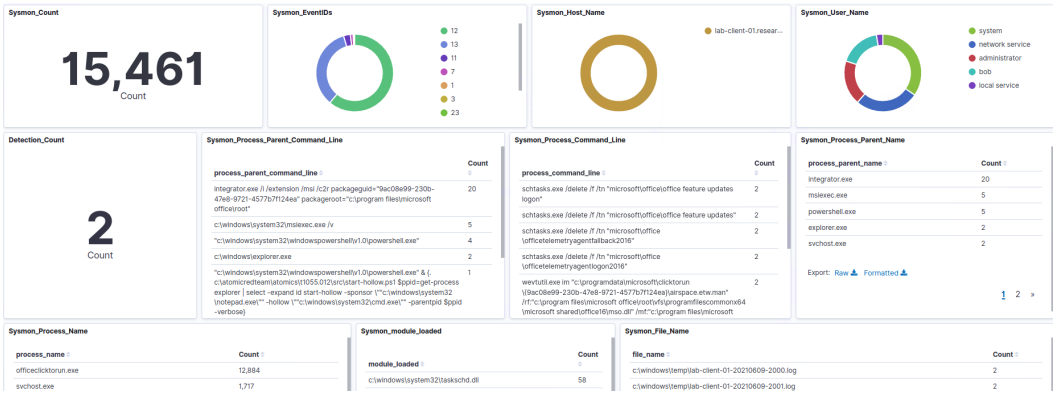

Figure 3. Kibana dashboard showing 100% detection rate from a set of 15,461 host events collected from one windows client using sysmon

pushes the data to `Logstash`. In `Logstash` the data is normalized and transformed, and the transformed data is sent to Elasticsearch. The data is converted to a JSON format and displayed on the Kibana dashboard.

The `Winlogbeat` stream is the most fundamental stream in the detection system, because it contains all the fields for each `Sysmon` event. All other streams, queries, joins and tables are derived from the `Winlogbeat` stream. Since HESPIDS is hierarchical in nature, the stream from `Winlogbeat` is the foundation of the hierarchy.

3.2. Tailored Sysmon Configuration

Sysmon [20] requires a configuration file when launching. This configuration file is customizable and events raised may be included or excluded as needed. The configuration used was based on a modified version of the Sysmon configuration developed by Malware Archeologist [2]. This Sysmon configuration was developed to maximize the detection capabilities while reducing the number of false positives; that is, events raised that do not correspond to malicious process injection. We further tailored Hartong's configuration to exclude related events raised by Microsoft Teams when launching and Google Chrome when running updates. This tailored Sysmon configuration file, in combination with our detection graphs, eliminated all of the false positives observed within our testing environment.

3.3. Process Injection Tools

To simulate repeatable process injection attacks two tools were chosen:

- Atomic Red [17]: Atomic Red is a library of simple focused detection tests, with few dependencies, that can be used by automation frameworks. The tests are small and portable and and mapped to the MITRE ATT&CK framework.
- InjectProc [15]: InjectProc is an open source project that allows automated testing of common process injection techniques including DLL injection, portable execution, and APC Injection.

4. Injection Experiments and Detection Graphs

Based on the article by A. Hosseini titled *Ten Process Injection Techniques* [7], we developed Table 3.3 that details those ten process injection sub-techniques, then maps these to their corresponding MITRE ATT&CK framework [16] sub-technique. Table 3.3 also maps the techniques, both as defined by Hosseini and by MITRE ATT&CK, to the tools we used for the injections, the corresponding research experiments we performed, and also the corresponding detection graphs. As shown in Table 3.3, Experiments 1 and 2 were performed using Atomic Red. Experiments 3, 4, and 5 were performed using InjectPro, with different call parameters.

Table 1 details the 11 sub-techniques for the Process Injection technique (T1055), as defined by MITRE ATT&CK, plus sub-technique 1056.04: Credential Hooking. Two sub-techniques apply to Linux hosts. Hence, 10 sub-techniques are applicable to Windows hosts. As stated before, and also shown in Table 1, in this research we analyzed five out of ten process injection sub-techniques in Windows. The five targeted techniques are shown in bold font in Table 1. We developed, implemented, and evaluated detection graphs for those five sub-techniques. For space reasons, we describe three of five detection graphs and corresponding evaluation experiments.

Table 2. Detail of process injection techniques as reported by hosseini [7] with mappings to mitre att&ck, performed experiments, and detection graphs

Hosseini ID	*Hosseini Sub-technique Name*	*MITRE ATT&CK*	*Experiment Number*	*Test Used*	*Detection Graph*
01	Classic DLL Injection	T1055.01	1	AtomicRed	Fig. 4
02	Portable Executable Injection	T1055.02	3	InjectProc	Fig. 6
03	Process Hollowing	T1055.12	2	AtomicRed	Fig. 5
04	Thread Execution Hijacking	T1055.03	None	N/A	N/A
05	Hook Inj. Via SetWindowsHookEx	T1056.04	4	InjectProc	Fig. 6
06	Inj. and Persist. Via Registry	T1546.09	None	N/A	N/A
07	APC Inj. and Atom Bombing	T1055.04	5	InjectProc	Fig. 6
08	Extra Window Memory Inj.	T1055.11	None	N/A	N/A
09	Injection Via Shims	T1546.11	None	N/A	N/A
10	IAT Hooking	T1056.04	None	N/A	N/A

The following subsections describe in detail three out of five selected experiments, the corresponding detection graphs, and the obtained results. It is important to note that none of the detection graphs, and their respective hierarchy of queries, include file names, nor process names, nor specific keywords as indicators of compromise. Instead, the detection is based purely on the correlation of events needed for malicious process injection to happen. This should make the detection much more resilient to changes on the implementation of the attack than currently used string matching based or machine learning detection approaches.

Before developing the detection graphs we dedicated a substantial amount of time analysing the host event data in search of patterns that would be suitable for a generic detection graph. Also, when designing and building our detection graphs we iterated through many versions. Once a detection query graph was converted into a set of hierarchical queries and then implemented in Kafka, using KSQL, the `Kafka` broker ran the query every minute on the incoming data streams. Then it forwarded results to `Logstash` where the output was normalized and forwarded to `Kibana`. The stream from `Kibana` was transformed into JSON and then forwarded to be displayed on the SOC analyst's dashboard. Figure 3 displays the value two, representing the number of positive hits for the top query after running one of the experiments.

4.1. Experiment 1: DLL Process Injection with Atomic Red

4.1.1. Experiment Details Technique: Process Injection: Dynamic-link Library Injection, T1055.001. **Atomic Test Name:** Process Injection via mavinject.exe. **Description:** Windows 10 utility to inject DLLs. Upon successful execution, powershell.exe will download T1055.dll to disk. Powershell will then spawn mavinject.exe to perform process injection in T1055.dll. With default arguments, expect to see a MessageBox, with Notepad's icon in taskbar.

4.1.2. Detection Graph Details Figure 4 illustrates the detection graph for Experiment 1: Classic DLL Injection. It contains five levels. Level one at the base of the hierarchy starts with the `Winlogbeat` stream. The final detection query resides at level five, at the top of the hierarchy. Moving up the hierarchy, the *WINLOGBEAT_REKEY_STREAM_PGUID* contains the parent process global unique identifier. The *WINLOGBEAT_REKEY_STREAM_GUID* is rekeyed on a processes' global unique identifier. Rekeying the `Winlogbeat` stream allows for renaming the fields, which allows for easier queries, and it allows for a set key for the streams at the next level in the hierarchy that are derived from either of these two streams. The key is used to join two streams together for query. Moving up the hierarchy to the middle level, there are three streams. These streams are *SYSMON_PROCESS_CREATION_PGUID*, *SYSMON_PROCESS_CREATION_GUID*, and *SYSMON_IMAGE_LOADED*. Each of these streams are constructed to detect specific malicious process injections. For testing purposes there were seven malicious process injection attacks, these three are the hierarchical basis for all attacks. At the fourth level resides *SYSMON_PROCESS_CREATION_TABLE_PGUID*. This table is created from a multi-join query. This table was created because KSQL documentation states that creating a multi-join query on streams only in a certain time frame can potentially lead to unpredictable

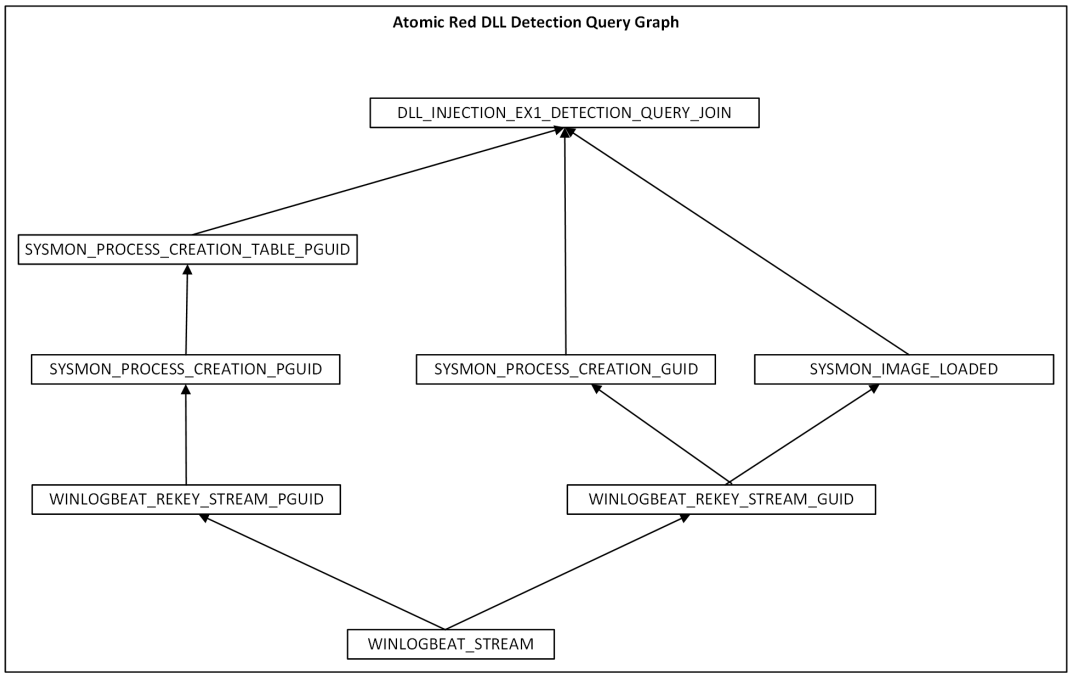

Figure 4. Detection graph for DLL injection with atomic red T1055.001

behavior. For this research it was determined that creating multi-join queries, within the same time frame, led to missed logs and detections. To solve this problem the *SYSMON_PROCESS_CREATION_TABLE_PGUID* was created. Once the table was created the logs were accurate with no missing logs and detections. The final detection query, at the top of the hierarchy, is *DLL_INJECTION_EX1_DETECTION_QUERY_JOIN*. This detection query is created using an inner join of *SYSMON_PROCESS_CREATION_GUID*, and *SYSMON_IMAGE_LOADED* with *SYSMON_PROCESS_CREATION_TABLE_PGUID*.

4.1.3. Detection Graph Results The detection graph successfully distinguished Atomic Test T1055.001 test number one via the sysmon-join-* index pattern in Kibana raising two hits. The detection query did not flag 16,244 events that are in the logs-* index pattern. The first hit shows that powershell.exe was used to start notepad.exe and then execute mavinject with the INJECTRUNNING switch on the process ID of the now running notepad process with T1055.001.dll. The second hit shows the actual target notepad process and pertinent information about the process.

4.2. Experiment 2: Process Hollowing with Atomic Red

4.2.1. Experiment Details Technique: Process Injection: Process Hollowing T1055.012. **Atomic Test Name:** Process Hollowing using PowerShell. **Description:** This test uses PowerShell to create a hollow from a Windows portable executable (PE) on disk with explorer.exe as the parent.

4.2.2. Detection Graph Details The detection graph for process hollowing, T1055.012 is built from the *DLL injection detection graph*, Figure 4. The graph for process hollowing (T1055.012) adds an additional stream at level three. This stream is derived from *WINLOGBEAT_REKEY_STREAM_GUID* and is named *SYSMON_PROCESS_HOLLOWING*. This new stream is specific to Sysmon event ID 25. This new stream is then inner joined with the *DLL_INJECTION_EX1_DETECTION_QUERY_JOIN* to create a new detection query named *DLL_INJECTION_EX2_DETECTION_QUERY_JOIN*. Figure 5 illustrates this detection graph.

4.2.3. Detection Graph Results The detection query successfully distinguished the simulated attack via the sysmon-join-* index pattern in Kibana raising two hits. The detection query did not flag 12,023 events that are in the logs-* index pattern. The first hit shows that powershell.exe was used to execute

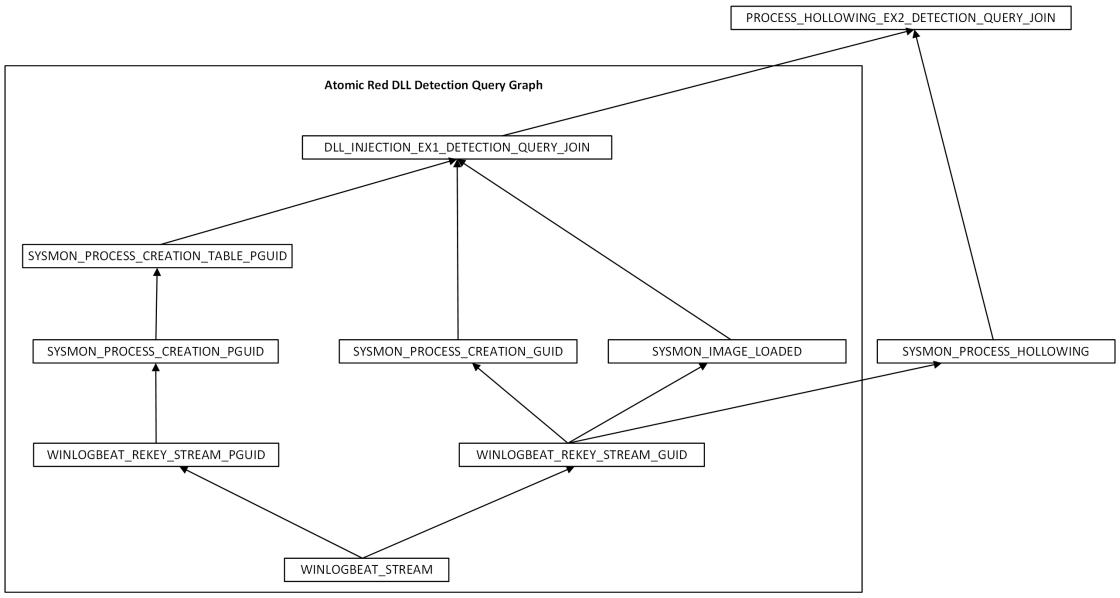

Figure 5. Detection graph for process hollowing with atomic red T1055.012

Start-Hollow.ps1 which hollows out and injects into the memory of a process defined in the script. The second hit shows the actual target process (cmd.exe) and pertinent information about the process.

4.3. Experiment 3: PE Process Injection with InjectProc

4.3.1. Experiment Details Technique: Process Injection: Portable Execution Injection, T1055.002. **InjectProc Test Name:** PE injection via InjectProc. **Description:** InjectProc defines the steps to inject the executable as: (1) Create target process and suspend it, (2) Unmap from memory, (3) Allocate space, (4) Write header and sections into the remote process, and (5) Resume remote thread.

4.3.2. Detection Graph Details Figure 6 illustrates the detection graph for Experiment 3: Portable Execution injection attack with InjecProc. Similar to Experiment 1 and Experiment 2, to detect InjectProc injection attempts the base of the hierarchical detection graph begins with the `Winlogbeat` stream. Moving up to hierarchy, the `Winlogbeat` stream is rekeyed to `WINLOGBEAT_STREAM_REKEY_GUID`, and `WINLOGBEAT_STREAM_REKEY_TUID`. It is important to note the stream names were purposely renamed from the Atomic Red attacks stream names. For example, the rekeyed stream for Atomic Red attacks was named *WINLOGBEAT_REKEY_STREAM_GUID*. For the InjectProc attacks the stream was named `WINLOGBEAT_STREAM_REKEY_GUID`, with the words *STREAM* and *KEYED* being swapped. This allowed the authors to easily determine which process injection tool was being used for the attack.

Moving up the hierarchy is the query `SYSMON_PROCESS_CREATION_GUID` which represents all the processes created. The query `SYSMON_PROCESS_ACCESS` detects all processes that are requesting access to another process. At the top of the hierarchy is the inner joined detection query `PROCESS_INJECTION_DETECTION_QUERY_JOIN`. This inner join is detecting all processes that were created that are trying to access another process. One should expect that this query would result in a fair amount of false positives. However, though this sequence of events happens in Windows systems, it does not happen very frequently. Our Sysmon configuration, in Subsection 3.2, prevented these from being marked as false positives.

4.3.3. Detection Graph Results The detection query successfully distinguished InjectProc PE Injection via the sysmon-join-* index pattern in Kibana raising four hits. Not flagged were 10,418 events from the logs-* index pattern. The first hit showed powershell.exe accessed the InjectProc.exe image. This hit also showed the executed injection command, the path to the injected process and the malicious DLL name used in the injection. The remaining hits show that other system executables accessed InjectProc.

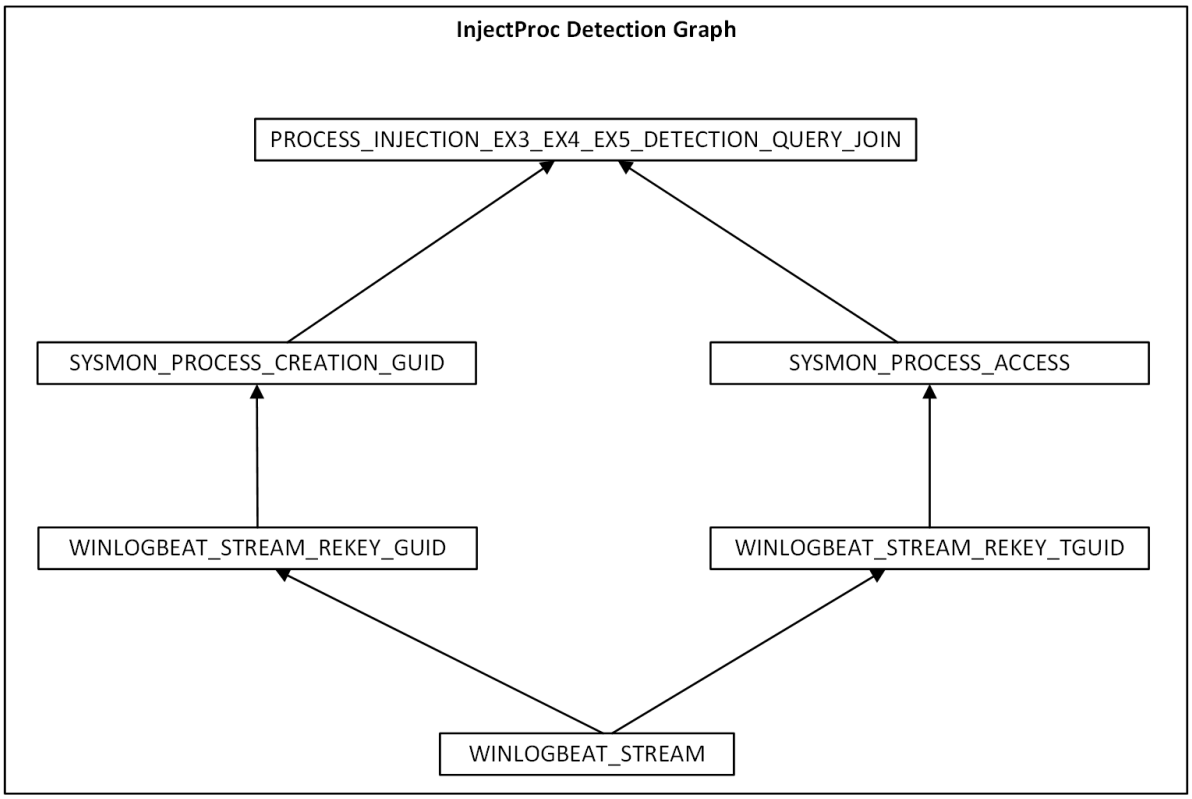

Figure 6. Detection graph for PE injection with injectproc T1055.002

4.4. Overall Results

Our graph-based HESPIDS approach was capable of detecting, with high accuracy, five out of eleven process injection techniques as detailed in table 3.3. We believe the HESPIDS graph-based detection approach is a step forward toward improving the state-of-the-art in the automated detection of advanced cyber attacks.

We believe that the current status quo is not a fault of the current state-of-the-practice but rather a lack of adequate state-of-the-art support for improved state-of-the-practice. It is not yet known how to appropriately specify nor mechanize complex threat hunting activities. Undoubtedly, an autonomous more powerful and mechanized form of correlation is needed to accurately detect LotL and process injection attacks. Furthermore, a common language for specifying and sharing high-level detection approaches is needed [21].

5. Conclusion and Future Work

Advanced malicious process injection techniques used by APT actors are very difficult to detect and mitigate in a timely manner. The problem of adequately detecting malicious process injection in Windows remains largely unsolved in the day-to-day operations. The graph-based HESPIDS approach presented in this article is a step toward a practical solution to this major challenge. We successfully designed and implemented a graph-based detection approach that is more resilient than all other known approaches. Furthermore, this approach was implemented using toolsets already in use today by the threat hunting and security communities.

For future work we envision the following: First, we plan to develop and implement detection graphs for the remaining types of process injection attacks. Second, we plan to expand the evaluation of our detection graphs by automating the generation of benign events. Third, we would like to scale the size and complexity of our evaluation environment.

Acknowledgments

We would like to give special thanks to Nate Guagenti (neu5ron) [6] for the many hours of invaluable help installing and configuring our tailored HELK mini-SOC environment and also to Roberto Rodriguez (Cyb3rWard0g) [18] for his very helpful and inspiring threat hunting blog posts. We would like to thank all the open source software stack and tools contributors.

We would like to thank the University of Idaho's, College of Engineering, Center for Secure and Dependable Systems, and Computer Science Department's technical and administrative staff for their help designing, implementing, and maintaining the research infrastructure used for this project. We would also like to thank the anonymous reviewers and mini-track chairs for their help improving this paper and the track and conference chairs for organizing the conference.

This research was partially funded by the U.S. National Science Foundation (NSF), under CyberCorps® award 1565572, the Idaho Global Entrepreneurial Mission (IGEM17-001), and the M.J. Murdock Foundation. The opinions expressed in this article are not those of the NSF, the M.J. Murdock Foundation, or the State of Idaho.

References

[1] Apache Software Foundation. Apache kafka. https://kafka.apache.org/, May 2021.

[2] Malware Archaeologist. The windows sysmon logging cheat sheet, Jan. 2020.

[3] Doug Burks. Security onion solutions. https://bit.ly/3z9f5ob, Jun 2021.

[4] Co. Elastic. Download winlogbeat. https://bit.ly/3zdZX9q, Jun 2021.

[5] FireEye. Highly evasive attacker leverages solarwinds supply chain to compromise multiple global victims with SUNBURST backdoor. https://bit.ly/3y7yEfB, Dec 2020.

[6] Nate Guagenti. Github: neu5ron. https://github.com/neu5ron, 2021.

[7] Ashkan Hosseini. Ten process injection techniques: A technical survey of common and trending process injection techniques. https://bit.ly/3sBe6L6, July 2017.

[8] Vasileios Mavroeidis and Audun Jøsang. Data-driven threat hunting using sysmon. In *Proceedings of the 2nd International Conference on Cryptography, Security and Privacy*. ACM, 2018.

[9] Microsoft. Group policy management console. https://bit.ly/3sA62dJ, Jun 2017.

[10] Microsoft. Enable attack surface reduction rules. https://bit.ly/3glqquf, Jun 2021.

[11] Joseph W. Mikhail, Jamie C. Williams, and George R. Roelke. ProcmonML: generating evasion resilient host-based behavioral analytics from tree ensembles. *Computers and Security*, 98, 2020. ISSN 0167-4048.

[12] Andrei Myllyla, Juuso & Costin. Reducing the time to detect cyber attacks: Combining attack simulation with detection logic. In *Proceedings of the 29th Conference of Open Innovations Association FRUCT*, pages 465–474, 2021.

[13] Matt Nelson. Reviving DDE: Using OneNote and Excel for code execution. https://bit.ly/2WfJH9e, Jan. 2018.

[14] Neo23x0. Github: Sigma rules hq. https://github.com/SigmaHQ/sigma, 2021.

[15] Secrary Noah and Nullbites. InjectProc. https://bit.ly/3z2mupr, Feb 2019.

[16] Anastasios Pingios, Christiaan Beek, and Ryan Becwar. MITRE ATT&CK: Process Injection. https://bit.ly/3B3YwdR, Feb 2021.

[17] Red Canary. Atomic red team. https://atomicredteam.io/, April 2021.

[18] Roberto Rodriguez. Cyber wardog lab. https://bit.ly/384Tp0C, Jun 2018.

[19] Roberto Rodriguez. Helk. https://thehelk.com/intro.html, Dec. 2020.

[20] Mark Russinovich and Thomas Garnier. Sysmon (v13.21). https://bit.ly/3ms2zgl, Jun 2021.

[21] Stuart Steiner, Ibukun Oyewumi, and Daniel Conte De Leon. MAHIVE. In *Proceedings of 2021 Hawai'i International Conference on System Sciences (HICSS-54)*, January 2021.

[22] Blake E. Strom, Joseph A. Battaglia, Michael S. Kemmerer, William Kupersanin, Douglas P. Miller, Craig Wampler, Sean M. Whitley, and Ross D. Wolf. Finding cyber threats with att&ck-based analytics. https://bit.ly/3kcxnii, Jun 2017.

[23] Yixin Sun and et al. Detecting malware injection with program-dns behavior. In *2020 IEEE European Symposium on Security and Privacy*, pages 552–568, 2020.